Using Intelligent Agents to Build Navigation Meshes

D. Hunter Hale, G. Michael Youngblood, and Nikhil S. Ketkar
University of North Carolina at Charlotte
Dept. of Computer Science, College of Computing and Informatics
9201 University City Blvd., Charlotte, NC 28223-0001
{dhhale, youngbld, nketkar}@uncc.edu

Abstract

We present a novel algorithm that allows agents to discover a navigation mesh for an environment as they move through the environment. The Navigation-Mesh Automated Discovery (NMAD) algorithm works by constructing its best guess for the navigation-mesh of a game level and then refines it when the agents moving through the world using the navigation mesh encounter unexpected or unknown obstacles. Using this algorithm, agents can enter a world in which they know nothing about, while still enjoying all of the advantages of a navigation mesh for path planning. We validated the effectiveness of this technique by showing that for both random and deliberative searches through multiple game worlds the error present in the best guess approximation the navigation mesh generated and maintained by NMAD converges to zero.

Introduction

How to represent a game or simulation world to an agent is one of the major decisions when using embodied agents in virtual worlds. A good representation will provide benefits to both the agents for navigation and reasoning, and possibly to other portions of the game engine. On the other hand, a poorly chosen representation can confuse or hinder agents attempting to traverse the virtual world (Tozour 2004). In recent years the navigation mesh has become the primary world space representation of choice for agents in virtual worlds (McAnils and Stewart 2008). The navigation mesh or *navmesh* is a world space representation that divides the walkable space in the world into some number of well-defined convex regions. The adjacencies of these convex regions of open space are then calculated and used to construct a graph of connectivity between convex regions. Each region is a node on the graph and if two regions share a common edge in space, they are connected by an edge.

Navigation meshes work well for agent navigation by allowing path finding algorithms to determine which regions hold the origin and destination points and then performs a search on the connectivity path to determine which regions to pass through. In addition, navigation meshes can assist with tasks beyond simple agent navigation. For example, collision detection can be accelerated by taking advantage of the navigation mesh. By locating an object in one of the open space regions of a navigation mesh it is assured that the object is not colliding with the world geometry. Furthermore, if the position of an object is known from the last frame it is highly probably that the object is still in the same region in the navmesh or in one of the neighboring regions and can quickly be located using a breath-first search starting at the object's last known position.

However, there are some problems with using a navigation mesh in a virtual environment. In order to generate the navigation mesh the configuration of the obstructions present in the world must be known in advance. This will obviously cause a problem if the target virtual environment is being procedurally generated at game initialization or even worse if the world is being procedurally expanded as players or agents move outside the existing world bounds. In addition, when agents are provided with a full navmesh at the initiation of a game or simulation it gives them full knowledge of the environment. This makes sense if the agent is moving through an area they would logically have traversed before (e.g., a guard in a building would be familiar with the building), but does not make sense for agents that should be ignorant of the areas they are entering (e.g., a rescue worker entering a building they have never been in before). Such extra knowledge can cause the agents to behave in ways a person would not behave and this reduces the believability of the agent.

These problems have prevented the use of navigation meshes in situations where the layout of the virtual world was unknown or unknowable prior to running agents in the world. We will present an algorithm that provides a solution to these problems and allows for the dynamic construction of a navigation mesh based on information gathered by agents in the world. We do this without sacrificing the benefits a navigation mesh provides to agents. At the same time agents are moving around through the world building and updating the navigation mesh they are querying and planning based on their current understanding of the navigation mesh. We accomplish this by using the Dynamic Adaptive Space Filling Volumes (DASFV) algorithm (Hale and Youngblood 2009a) combined with sensors that are integrated into each agent to detect incorrect classifications of space in the navigation mesh versus the actual world geometry. In this paper we will present a brief overview of DASFV and then a new algorithm which builds of the DASFV framework to allow agents to discover the navmesh for the environment they operate in.

Related Work

We are not aware of anyone doing previous work in attempting to dynamically learn navigation meshes using agent movement in a virtual environment. However, the task of map learning (Thrun et al. 1998; Kuipers and Byun 1987; Youngblood, Holder, and Cook 2000; Millán and Torras 1992; 1992; Yamauchi 1998) in robotics is intimately related to the task of dynamically constructing and updating navigation meshes. Map learning involves one or more robots exploring a physical environment and building an abstract representation of the environment. The similarity between map learning and dynamically generating navigation meshes is that both the tasks involve building an abstract representation, which will allow for better path planning in future. The critical difference is that in the case of map learning, the robots are limited by their sensors and cannot accurately perceive the exact position and the dimensions of the objects they have collided with. This makes the problem of map learning much more challenging than dynamically building a navigation mesh as there is an inherent uncertainty and noise in the percepts.

Methodology

The primary contribution of this paper is the Navigation-Mesh Automated Discovery (NMAD) algorithm, which allows agents to discover the navigation mesh for a game level while traversing the level. This algorithm works by making an initial assumption that the world is empty and generating an appropriate navigation mesh for this empty world. As the one or more agents move through the world, each agent will detect and report any obstructing geometry (objects) they encounter. If this newly discovered geometry is not present in the navigation mesh then the Dynamic Adaptive Space Filling Volume (DASFV) algorithm will update the mesh. As these agents move through the world discovering geometry, the current decomposition (a decomposition is a breakdown of obstructed and open space in the game world in a navigation mesh) will eventually converge on the ideal decomposition.

Dynamic Adaptive Space Filling Volumes

The Dynamic Adaptive Space Filling Volumes (DASFV) algorithm is a two part algorithm (Hale and Youngblood 2009a). The first part is a form of the Adaptive Space Filling Volume Algorithm (either Planar Adaptive Space Filling Volume (PASFV) or Volumetric Adaptive Space Filling Volume (VASFV)) depending on whether DASFV is being executed on a 2D representation of a game world (PASFV) or the actual 3D geometry of the world (VASFV). This first part of DASFV generates an initial navigation mesh in the following manner.

Both PASFV and VASFV initially seed a grid of unit sized regions, quads in the case of PASFV or cubes for VASFV, into the open space in the world. These regions are then provided an iterative chance to expand each of their faces one unit in the direction of that faces normal. When either type of region encounters an obstruction (either another growing region or world geometry) if the region is perfectly adjacent to the obstruction it will stop growing in the direction of that face. However, if the growing region is not perfectly adjacent to the obstruction the algorithm calls for the insertion of another face into the region to better approximate the obstructing geometry. The insertion of extra faces allows the quads of PASFV to become five sided or higher order polygons, and the cubes of VASFV to become seven sided or higher order polyhedrons. After all the regions in the world have grown to the maximum possible extent, the algorithm will attempt to place more unit quads or cubes into the world to fill any open space adjacent to existing regions. These new regions then grow and expand outward until they collide with obstructions and once all of them have stopped growing they are allowed to generate more new regions. This cycle of grow as much as possible and then placement of new regions continues until no more regions can be placed, at which point the world will be fully decomposed. For more details on these algorithms please see (Hale, Youngblood, and Dixit 2008) and (Hale and Youngblood 2009b)

The second half of the DASFV algorithm uses the navigation mesh generated by either PASFV or VASFV as a starting point and DASFV modifies this navigation mesh in response to changes in the simulation environment at runtime. As the underlying geometry of the game or simulation environment is altered (e.g., walls are demolished or buildings collapsed into formally open areas) the navigation mesh is also updated in real time to reflect the current state of the world. It accomplishes these updates by first adding or removing geometry to reflect the new game environment. Then it rebuilds the decomposition of the world by placing new regions to fill any holes in the navigation mesh and running either PASFV or VASFV. Once these new regions have been placed in the world and allowed to grow and seed fully, the connectivity between affected regions can be regenerated to produce an effective navigation-mesh. Using DASFV it is possible to generate a high quality navigation mesh for dynamic environments that can be applied to multiple uses in a game or simulation world. These updates are made assuming perfect knowledge of the world, and because of this they do not reflect the restricted knowledge agents in the world should have based on their sensory limitations.

Navigation-Mesh Automated Discovery

Navigation-Mesh Automated Discovery is an extension of the DASFV algorithm to represent the limited knowledge each agent or group of agents' posses about their local environment. The NMAD algorithm as shown in Algorithm 1 begins by initializing an empty navigation mesh composed of a single region that covers all possible negative (walkable) space areas present in the world. It also incorrectly classifies all of the positive (obstructed) space areas present in the world as negative space. While this navigation mesh is inaccurate, it is the most accurate navigation mesh it is possible to create with no knowledge about the world. The accuracy of the navigation mesh will improve with the addition of positive space locations discovered by the agents.

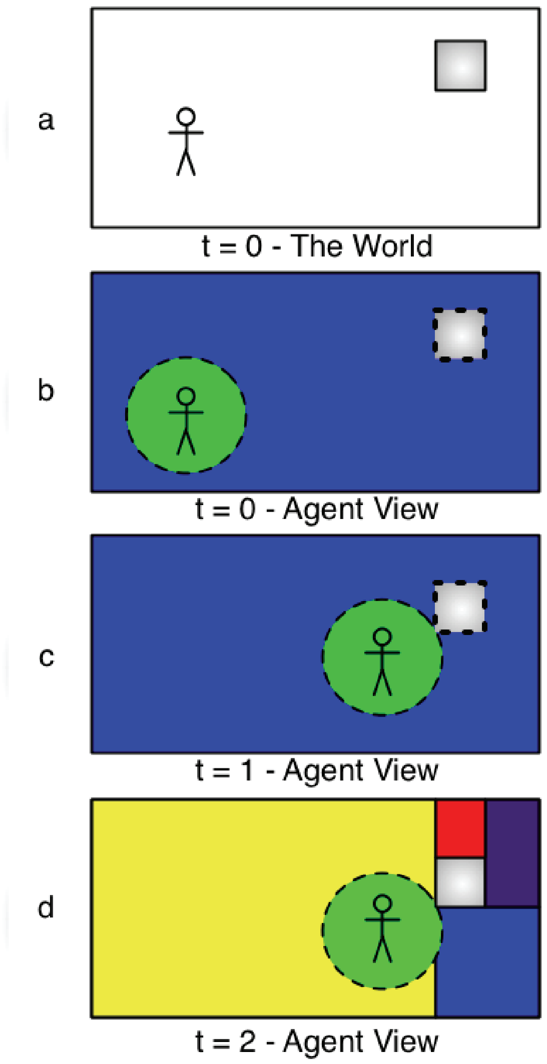

Figure 1: This collection of images shows a sample agent traversal and discovery of positive space. In image (a) we see the actual state of the world. The agent is present in the lower left corner and there is a single positive space obstruction (shaded gray) in the upper right. (b) shows the initial view of the world as the agent perceives the world through the navigation mesh. The agent's detection area is shown as the (green) circle. Since no negative space objects have been discovered, the navigation mesh is a single region covering the entire world (blue). (c) shows the agent moving and discovering a positive space obstruction. At this point the obstruction will be added to the navigation mesh. (d) shows the status of the navigation mesh after the addition of the positive space region discovered in (c).

Once the initial one region navigation mesh has been constructed, the agents present in the world can use it for navigation as shown in Figure 1a and 1b. Each agent present in the world is modeled as having a individually defined detection range to notice positive space obstructions. As these agents move through the world, they will encounter positive space obstructions as shown in Figure 1c.

When a new positive space region enters the detection range of an agent then one of two things will happen depending on how the world is represented. If the positive space objects were constructed in a monolithic manner, (e.g., one positive space object represents an entire building) then the positive space object will need to be sliced into smaller components. We do this slicing because the detection range of an agent is representative of the agent's ability to see the world, and it does not make sense that by seeing one corner of a building the agent would become aware of the entire extent of that building. This can either be done in advance by creating preset splits in the object or at runtime by carving off sections of an obstruction via polygonal subdivision. However, if the positive space objects are not constructed in a monolithic manner and instead are created from smaller building blocks then the blocks can be directly consumed by NMAD without subdivision. For dynamically generated worlds the positive space objects list will be derived from the components used to create the world and can be subdivided based on how they would be rendered.

Once a positive region has been identified then DASFV is used to insert the new positive space into the existing navigation mesh with the minimal possible disruption to the navigation mesh as shown in Figure 1d. DASFV works by first locating areas of negative space that intersects the area of the positive space we are adding and then removing them. The positive space region is then inserted into the navigation mesh. The negative space regions adjacent to the ones that were removed are then allowed to reseed the world with more regions to fill the newly vacated areas. These newly placed regions then grow as much as possible and can generate more regions if needed to ensure that all of the newly vacated space is fully decomposed.

The algorithm then waits for another positive space region to be detected. Even if all of the positive space regions present in the world have been detected, the algorithm can continue to run at which point it becomes a form of localized DASFV as only changes in the world geometry that pass within the agents detection area will be reflected on the navigation-mesh. This means that every agent on the map is not instantly aware of a new door being created in a wall or a passageway being closed off by rubble.

Finally we have made three improvements to the base NMAD algorithm. First, when no changes are detected in the world the navigation mesh quality can be improved by combining adjacent regions when the resulting new region would be still be convex. Doing this results in a smaller, more compact navigation mesh while amortizing the cost of these improvements across multiple agent update cycles. This reduces the search space present in the navigation mesh and thereby speeds up queries. Secondly, more than one agent can feed information into the NMAD algorithm. It is

Algorithm 2: The NMAD algorithm loop

```
/* The NMAD algorithm is generally
   called from inside the agent
   class.  and is assumed to have
   access to all of the agent local
   variables */
/* Check to see if any unfound
   positive space objects are within
   range of the agent */
$List\ FoundPositiveSpace$;
for $NewPosSpace$ in World do
    if Agent.CanSee($NewPosSpace$) then
        $FoundPositiveSpace$.add($NewPosSpace$);
if $FoundPositiveSpace$.size() != 0 then
    /* We found new positive space
       areas */
    for $NewPosSpace$ in $PositiiveSpaceList$ do
        /* Insert the positive space
           into the navmesh */
        $NavigationMesh$.insert($NewPosSpace$);
    /* Regrow the affected areas with
       DASFV */
    $NavigationMesh$.regrow();
    /* Rebuild the connectivity */
    $NavigationMesh$.reconnect();
else
    /* Clean up the navmesh instead */
    $NavigationMesh$.cleanup();
```

possible for many agents to simultaneously search for positive space obstructs rather than one and by exploring the world with multiple agents it is possible to converge on a perfect decomposition faster. Finally, two or more separate navigation meshes can be maintained along with separate groups of agents who query and update just their own navigation mesh. This produces the effect of creating two or more teams of agents each with their own unique understanding and view of the game environment.

Experimentation

We performed a series of evaluations to asses whether the NMAD algorithm correctly built a navigation mesh based on an agents observations of the surrounding positive space obstructions. To perform these experiments we constructed a sandbox test environment. This environment consisted of 7225 meters square (85m * 85m) of open space that the agent could traverse. This is roughly the size of two football fields placed side by side for comparison. This world was then randomly populated with obstructions. The obstructions were composed of three different sizes of cubes (1 meter, 2 meter, 5 meter). The cube placement was restricted such that two cubes cannot overlap each other. An agent was then placed into the world. This agent had a detection radius of 10 meters for obstructions. The test agent moved at a rate of 1.25 meters per second. At this speed traversing the level along an edge would take would take 68 seconds.

The agent randomly chose destinations in what it believed to be unoccupied space somewhere in the level. Navigation to these randomly selected points was controlled by two distinct methods. The first local navigation method was used when both the target location and the agent were located in the same region of the navigation mesh. In this case, the agent will move in the direction of the destination at a normal walking pace. The second form of navigation option occurs when the target point location and the agent are in different regions of the navigation mesh. In this case, the agent searches the navigation mesh to locate a path from its current location to the goal region. This path is then stored and the agent will move through the centers of the shared edges between connected regions on this path. When the agent is in the same region as its target and it enters local navigation mode. It is worth mentioning that for simplicity of implementation the agent uses a breadth first search to find a path rather than a more complicated best first search algorithm.

Our agent implemented the NMAD algorithm presented in this paper to update the navigation mesh as it moved through the world. We performed 10 passes through this world where we measured the error present in the navigation mesh in the form of incorrectly classified regions. One of these passes through the world is illustrated in Figure 4. We assumed that all of the negative space and discovered positive space would be correctly decomposed because the basic algorithm underpinning NMAD generates near perfect coverage (except for completely disconnected regions which will not be represented in the navigation mesh) spatial decompositions to use as navigation meshes. Recall that initially, and until it learns otherwise, the NMAD algorithm considers all unknown space to be negative space. This means that the only incorrectly classified regions would be positive space regions that the agent has not yet discovered. Initially, since the agent starts with no knowledge of its surroundings all of the geometry in the world was incorrectly classified giving our algorithm a one hundred percent misclassification rate at time 0. We then measured the misclassification percentage at ten second intervals while the agent wandered the world on each of the 10 passes we performed.

The results of this experiment are presented in Figure 2. The incorrect classification quantity converged on zero quickly taking on average 225 seconds.

After conducting ten walks through the same world using a random movement agent, we then implemented an agent who performs a spiral pattern from the center of the world outward. This agent running the NMAD algorithm was then allowed to traverse five randomly generated worlds. These worlds contained between six and twelve randomly placed and sized obstructions. In addition, the agent's speed was increased to 3.6 meters per second in order to reduce the time required to run this experiment. The graph shown in Figure 3 shows the results of this experiment. The agents spiral pattern was designed such that the entire world would pass through the agent's visibility radius so if NMAD is working correctly all space in the world will be correctly classified. From this graph, we see that NMAD will discover and correctly classify all of the space in the world within 90

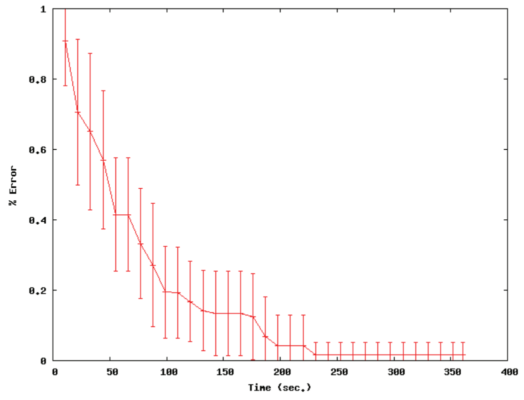

Figure 2: A graph showing the convergence of incorrectly classified positive space regions to zero over time averaged accross 10 random walks through the world by an agent. The y-axis gives the error as a percentage of incorrectly classified positive space in the world while the x-axis shows the traversal time. The bars on the graph provide the standard deviation across the multiple agent traversals; the standard deviation is rather large because the random nature of the agents movement can quickly discover all the positive space in the world or rather slowly.

seconds.

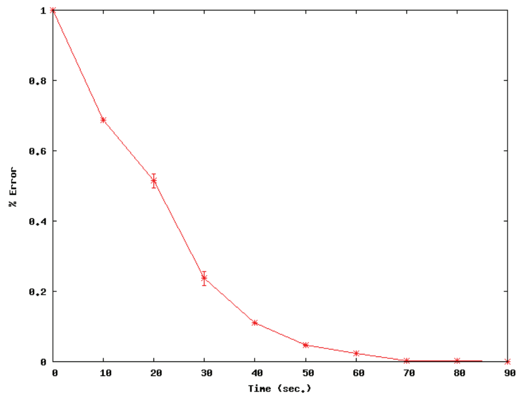

Figure 3: A graph showing the convergence of incorrectly classified space to zero over 5 spiral searches through randomly generated worlds. The percentage of misclassified positive space present in the world is shown on the y-axis while the x-axis shows the agents travel time in seconds.

The primary purpose of these two experiments is to show that the NMAD does correctly classify all of the space present in a game world. In both the random pathing and the random world generation experiments the misclassification in the navigation mesh eventually reached zero. By converging to zero error across many different traversals of the world, we show that the algorithm does consistently and reliably classify all of the space in the game world.

Conclusion

Our Navigation-Mesh Automated Discovery algorithm provides an effective way to generate navigation meshes in worlds where the geometry of the environment is not known in advance. By dynamically creating a navigation mesh based on information discovered by agents traversing the virtual world, we are able to produce navigation meshes for worlds that previously have been impossible to define in advance (e.g., procedurally generated worlds). The navigation meshes provide an agent with information from the moment the simulation or game starts and will continue to improve as agents move through the world. By supporting multiple agents or different groups of agents we provide a method for a team of agents to explore an area or to have two or more agents each maintain their own view of the world. In short, by using the NMAD algorithm navigation meshes can be generated on demand without any knowledge of the world they represent and improved as agents move through the world.

Acknowledgments

This material is based on research sponsored by the US Defense Advanced Research Projects Agency (DARPA). The US Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the US Government.

References

Hale, D. H., and Youngblood, G. M. 2009a. Dynamic updating of navigation meshes in response to changes in a game world. In *Florida Artificial Intelligence Research Society Conference*.

Hale, D. H., and Youngblood, G. M. 2009b. Full 3D Spatial Decomposition for the Generation of Navigation Meshes. In *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*.

Hale, D. H.; Youngblood, G. M.; and Dixit, P. 2008. Automatically-generated Convex Region Decomposition for Real-time Spatial Agent Navigation in Virtual Worlds. In *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*.

Kuipers, B., and Byun, Y. 1987. A qualitative approach to robot exploration and map-learning. In *Spatial Reasoning and Multi-Sensor Fusion: Proceedings of the 1987 Workshop, October 5-7, 1987, Pheasant Run Resort, St. Charles, Illinois; Sponsored by AAAI*, 390. Morgan Kaufmann.

McAnils, C., and Stewart, J. 2008. *AI Game Programming Wisdom 4*. Charles River Media. chapter 2.4 Intrinsic Detail in Navigation Mesh Generation, 95 – 112.

Millán, J., and Torras, C. 1992. A reinforcement connectionist approach to robot path finding in non-maze-like environments. *Machine Learning* 8(3):363–395.

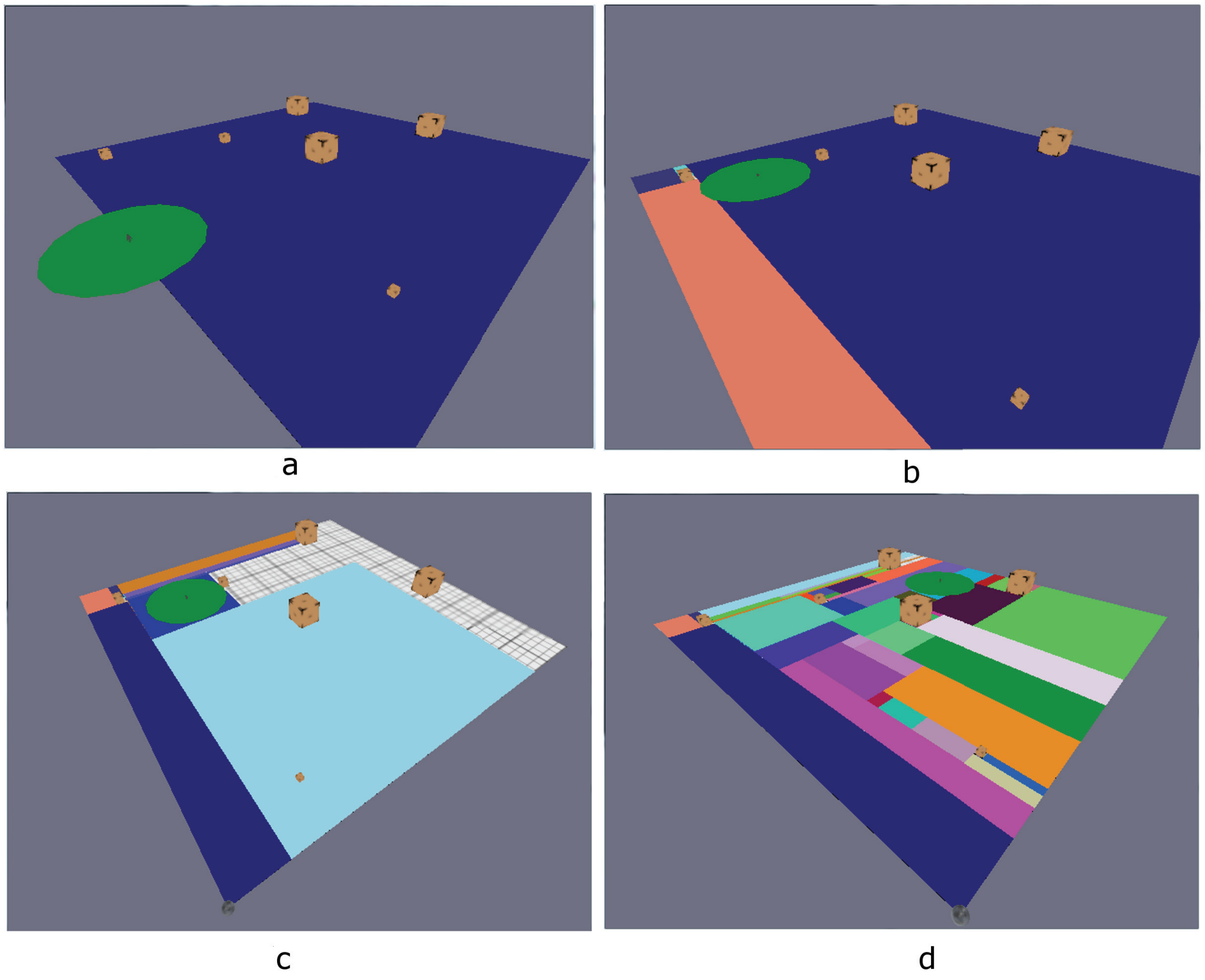

Figure 4: This collection of images was taken during one of the test runs of the algorithm. The agent is the small gray figure in the middle of the green circle. The green circle represents the range of the agent's ability to detect errors in its navigation mesh. The randomly colored areas in the figure are regions of the navigation mesh. The obstructions are visible as boxes sitting above the decomposition. Image (a) shows the initial state of the algorithm. In this image the navigation mesh (blue) assumes that the entire world is walkable and can be represented as a single region. In Image (b) the wandering agent has encountered the first positive space obstruction. Image (c) shows NMAD as it is the process of building new regions for the navigation mesh. The grid texture is negative space that has not yet been claimed by any decomposition. Normally this section of the algorithm would run almost instantly but the it was intentionally slowed down to capture this image. Image (d) shows the final navigation mesh after all of the positive space obstructions have been discovered and correctly classified.

Thrun, S.; Bucken, A.; Burgard, W.; Fox, D.; Frohlinghaus, T.; Hennig, D.; Hofmann, T.; Krell, M.; and Schmidt, T. 1998. Map learning and high-speed navigation in RHINO. *AI-based Mobile Robots: Case studies of successful robot systems. MIT Press, Cambridge, MA.*

Tozour, P. 2004. *AI Game Programming Wisdom 2*. Charles River Media. chapter 2.1 Search Space Representations, 85–102.

Yamauchi, B. 1998. Frontier-based exploration using multiple robots. In *AGENTS '98: Proceedings of the second international conference on Autonomous agents*, 47–53. New York, NY, USA: ACM.

Youngblood, G. M.; Holder, L. B.; and Cook, D. J. 2000. A framework for autonomous mobile robot exploration and map learning through the use of place-centric occupancy grids. In *ICML Workshop on Machine Learning of Spatial Knowledge*.